Licenciatura em Engenharia Informática e Multimédia

Modelação e Simulação de Sistemas Naturais - MSSN

Semestre 2021/2022

Relatório Projeto Final

Docente: Eng.º Paulo Vieira

Trabalho Realizado por:

Fábio Dias, 42921

Índice

Indice de Imagens

Introdução

Neste projeto final, teve-se como objetivo a interactividade entre o utilizador e a resposta do programa. Para isso, foi necessária uma ideia exequível, minimamente interessante e intuitiva.

Dado que um dos temas que mais me interessa é o desenvolvimento de videojogos, que cumpre os objetivos de interatividade do utilizador e a resposta do programa, decidi realizar uma pequena mecânica que reage à habilidade do utilizador. Para isso, foi fulcral a utilização de uma simulação de *Diffusion Limited-aggregation* (D.L.A.).

Desenvolvimento

Comecei por estabelecer o que seria o conceito para este projecto. Um círculo que se desfaz em pequenas partículas e que, quando essas partículas se juntam, é refeito. Estas partículas possuem uma velocidade e uma direção que são influenciadas pela habilidade do jogador. Defini estas partículas como *Walker's*, um dos conceitos aprofundados durante o semestre, e defini também uma classe `Center`, que se trata do círculo de matéria central.

De forma a conseguir a interatividade, estabeleci que esta separação e união das partículas aconteceria dado o *input* do jogador. Quando o jogador carrega com o lado esquerdo do rato, separar; quando carrega com o lado direito, unir. Para evitar repetição de comportamentos, implementei um tempo de espera após o input do jogador. Isto permite que o comportamento tome efeito durante um pequeno período de tempo, até o utilizador poder ter controlo no programa novamente.

Para tornar o programa mais interessante e não restringir tanto o controlo do jogador, permiti que a união destas partículas fosse cancelada a meio do comportamento, assim como a sua separação. Desta forma, é possível experienciar uma simulação mais dinâmica.

O factor que influencia o comportamento das partículas é dado pela habilidade do jogador. Para tal, criei uma barra lateral que possui um indicador que se move para ambos os extremos verticais da barra. Esta barra é dividida em cinco secções: uma verde no centro, duas vermelhas nas extremos e duas laranjas entre o centro e os extremos. Ao carregar com o lado esquerdo do rato, este indicador pára em cima de uma destas secções e torna o comportamento das partículas mais errático, quanto mais aos extremos estiver, ou mais certo e sublime, quanto mais ao centro estiver.

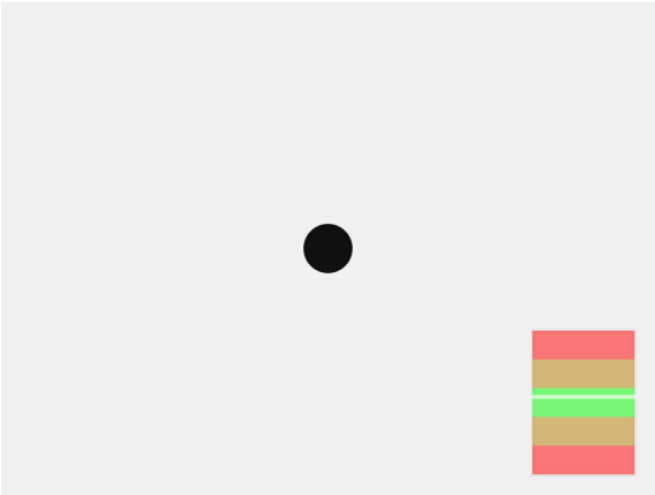

Figura 1 - Simulação com as partículas aglomeradas

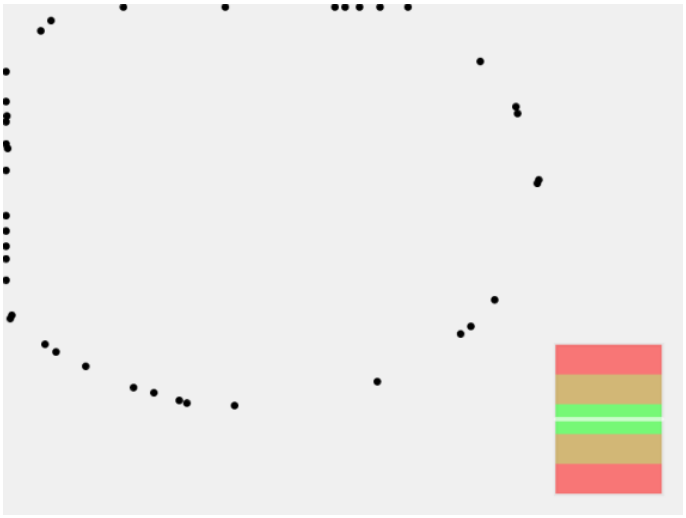

Figura 2 - Simulação com as partículas a divergir

Código

`Center`

Como anteriormente referido, para o círculo central, criei um classe `Center`, cujo constructor estabelece a sua posição. Possui os métodos `SetRadius`, que configura o raio do centro, `UpdateRadius` que atualiza o raio e `Display`, que, tal como todos os outros métodos semelhantes, faz o tratamento da aparência do objeto.

```
public class Center {

    private PVector pos;

    private float radius;
    private boolean isMaximumSize;

    public Center(float x, float y)

    public void SetRadius(int numberOfWalkers)

    public void UpdateRadius(List<Walker> walkers)

    public void Display(PApplet app)
}
```

Figura 3 - Código para a classe Center

`Walker`

Tal como o `Center`, esta classe possui um constructor que estabelece a sua posição e também informa que é para parar no centro do ecrã, assim como estabelece o seu estado actual como parado. Tem os seguintes métodos:

1. `Wander`, que o permite mover-se, sempre a ser puxado para o local definido pelo estado do programa;
2. *`UpdateState`* que gere o seu estado;
3. `SetOuterDirection` que define uma posição na direcção às bordas da janela;
4. `SetInwardDirection` que define a direcção desejada como sendo o centro da janela;
5. `SetSize` define o seu raio;
6. `SetSpeed` define a sua velocidade;
7. `SetStepLerp` define a intensidade com que este vai ser puxado para o posição pretendida;
8. `SetCenterThreshold` que define a partir de que posição está o centro;

9. `SetCenter` para obter a referência ao centro;
10. `GetPos` devolve a posição actual;
11. `GetWalkerState` que devolve o seu estado;
12. `Display`.

```
public class Walker {

    private Center center;

    public enum WalkerState {

    public PVector pos;
    private PVector direction;

    private WalkerState currentState;

    private boolean isToStopAtCenter;
    private int color;
    private float radius = 4;
    private float movementSpeed = 0.25f;
    private float stepLerp = 0.00002f;
    private float centerThreshold = 0.25f;

    public Walker(PApplet app)

    public void Wander(PApplet app)

    public void UpdateState(PApplet app, List<Walker> walkers)

    public void SetOuterDirection(PApplet app)
    public void SetInwardDirection(PApplet app)

    public void SetSize(float radius)

    public void SetSpeed(float movementSpeed)

    public void SetStepLerp(float stepLerp)

    public void SetCenterThreshold(float centerThreshold)

    public void SetCenter(Center center)

    public PVector GetPos()

    public WalkerState GetWalkerState()

    public void Display(PApplet app)
}
```

Figura 4 - Código para a classe Walker

GameApp

É esta classe que gere o programa. Implementa a interface `IProcessingApp`. Possui uma referência ao centro e uma lista de *Walker's*. Possui, para além disso, três atributos: `maxWalkers`, o número máximo de Walkers a serem inicializados; o `insideCenter` que é o número de *Walker's* que se encontra dentro do centro; e o `stepsPerFrame` que define quantos passos são dados na simulação do comportamento dos *Walker's* - acelerando ou desacelerando a simulação. Possui

também dois enum's: `State`, que define em que estado se encontra o objectivo dos *Walker's*: `TOGETHER` ou `SPREAD`. E `SliderState`, que controla em que direcção o indicador se está a mover, `UP` ou `DOWN`. Assim como duas variáveis de controlo destes enums, `currentState` e `currentSliderState`.

Como mencionado anteriormente, de forma a obter um intervalo de tempo entre os estados do programa, criei duas variáveis, `timeBetweenStates` que define o tempo entre eles e o `betweenStatesTimer` que conta o tempo que passou desde que houve uma transição no estado do programa.

Para parar o indicador, usei a variável boleana `canSliderMove` e uma variável para definir a sua posição na vertical, `sliderYPos`.

No método `setup` inicializei o `Center` e os *Walker's*, defini o estado inicial do programa, o tempo de espera entre estados e o número máximo de *Walker's*. No método `draw` decremento o tempo entre os estados, caso seja necessário; é simulado os passos dos *Walker's* e actualizado o seu estado, atualizo o raio do `Center` e desenho todas as entidades mencionadas. Desenho a barra e o apontador pelo método `Slider` e ainda desloco o apontador, caso consiga. Para além de desenhar, o método `Slider` também define se o apontador deve mudar a direcção em que se desloca.

No método `mousePressed`, identifico qual o botão do rato que foi accionado e, caso seja possível, altero o estado do programa. Também, se for possível, altero os atributos dos *Walker's* pelo `SetWalkerStats` que dá uso aos métodos dos *Walker's*.

```
public class GameApp implements IProcessingApp
{
    private Center center;

    private List<Walker> walkers;
    private int maxWalkers;
    private int insideCenter;

    private int stepsPerFrame = 100;

    public enum State

    private State currentState;

    private float timeBetweenStates;
    private float betweenStatesTimer;

    private enum SliderState

    private SliderState currentSliderState = SliderState.DOWN;
    private boolean canSliderMove = true;
    private float sliderYPos = 400;
```

Figura 5 - Código para a classe GameApp

```
    public void setup(PApplet parent) {
    public void draw(PApplet parent, float dt) {
    public void keyPressed(PApplet parent) {
    public void mousePressed(PApplet parent) {
    private void Slider(PApplet app)
    private void SetWalkerStats()
}
```

Figura 6 - Código para a classe GameApp

Diagrama de Classes

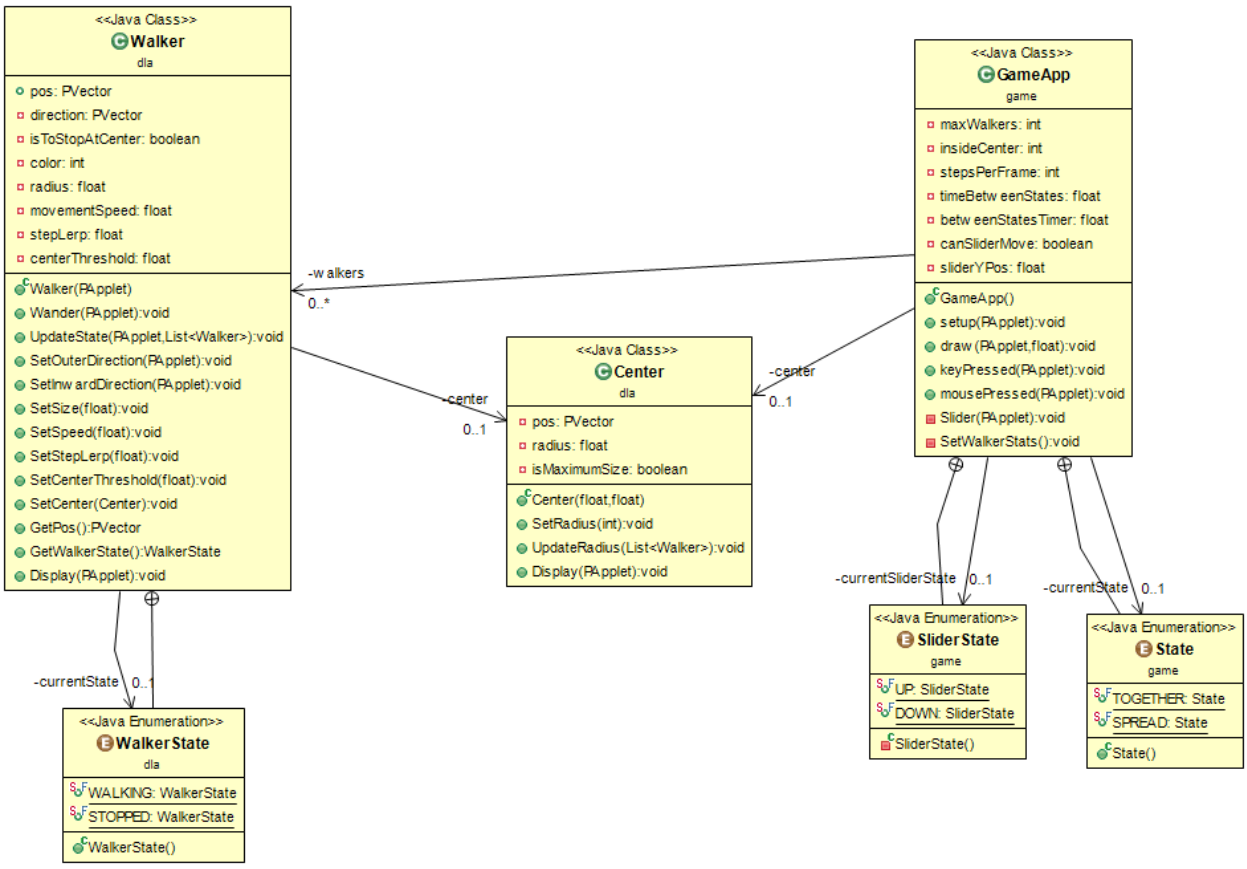

Figura 7 - Diagrama de classes da aplicação

Conclusão

Assim, é possível concluir que o estudo dos D.L.A. foi crucial para a implementação deste projeto e para a simulação de partículas que se pretendia. Foi possível concretizar um projeto intuitivo e com um certo grau de aleatoriedade que permite a simulação de diferentes cenários a cada iteração.